Instituto Tecnológico y de Estudios Superiores de Monterrey

ESCUELA DE INGENIERÍA Y CIENCIAS

Inteligencia Artificial Avanzada para la Ciencia de Datos I

Manual de Usuario - Implementación Manual ML: Regresión Lineal Simple.

Presenta:

Miguel Ángel Pérez Ávila

Profesor:

Dr. Víctor Manuel de la Cueva Hernández

Sante Fe, Ciudad de México a 21 de Agosto del 2025

Índice

Introducción

El presente documento presenta la implementación y metodología de ejecución de un programa escrito en Python que realiza procesos afines al entrenamiento de una regresión lineal simple calculada mediante el ajuste de los parámetros θ mediante la optimización del error utilizando el método de Gradiente Descendiente. Se utilizó la técnica de vectorización en la medida de lo posible para realizar los cálculos involucrados. Las funciones implementadas incluyen:

- La lectura de un archivo de entrada
- La ejecución del gradiente descendente aplicado a vectores X, Y y θ dados, considerando un paso Alpha a través de una cantidad de N épocas/iteraciones.
- El cálculo de la función de costo J(θ) para un vector X, Y y θ dados.
- La graficación de la dispersión de ciertos vectores X, Y dados junto con la recta ajustada mediante un vector de θ dados.

Implementación

Es importante mencionar que para le ejecución de este programa es necesario contar con las siguientes librerías en el entorno de ejecución y una versión de python igual o cercana a la utilizada para esta implementación siendo esta la 3.11.13:

- **numpy** (Manipulación de vectores y operaciones matemáticas en general).
- **matplotlib** (Graficación y visualización de datos).
- **chardet** (Detección de codificación para la lectura de los archivos de entrada).

En caso de no contar con alguna de las anteriores se pueden instalar y/o actualizar en bash mediante:

```
// En caso de utilizar conda
user@pc: conda activate <dev_env>

user@pc: pip install numpy matplotlib chardet
```

Se utilizó un enfoque orientado a objetos para la implementación de la regresión lineal en su totalidad, creando una estructura/clase llamada MLDev que proporciona las herramientas necesarias para realizar el cálculo de la regresión.

MLDev implementa la siguiente estructura explicada en términos de tipo de dato abstracto:

- Atributos de la estructura:
    - **X** → Matriz de características (incluye columna bias).
    - **Y** → Vector de etiquetas.
    - **Costos** → Historial de costos en gradiente descendente.

- Interfaz de la estructura:

- Carga y preparación de datos
    - **getFileEncoding(path)** → Detecta la codificación del archivo de datos.
    - **readFile(path, separador)** → Lee el dataset, construye las matrices X (features con bias incluido) y Y (etiquetas).
    - **setXVector(X)** → Define manualmente el vector/matriz de características.
    - **setYVector(Y)** → Define manualmente el vector de etiquetas.

- Entrenamiento de modelos
    - **gradDesc(n_epochs, alpha, thetas)** → Calcula los parámetros óptimos mediante gradiente descendente.
    - **calcCost(X, Y, thetas)** → Calcula la función de costo J(θ).

- Visualización
    - **graph(x, y, xlabel, ylabel, title)** → Gráfica de curvas como evolución de error vs. épocas.
    - **scatterPlotAndLine(X, Y, thetas)** → Gráfica de dispersión de datos y la recta ajustada.

Mediante la estructura anterior se implementan las siguientes funciones utilizadas para este proyecto:

```
# Función para graficar la dispersión de datos (Dados los Vectores X y Y) y la función Recta ajustada
# Está función ASUME que el vector de X aún contiene la columna X0 del bias con "unos"
# - Input:
#       - x: Vector de X (Aún conteniendo la columna de X0)
#       - y: Vector de Y
#       - theta: Vector de thetas óptimas para la recta
# - Return : Void
def graficaDatos(x, y, theta)

# Función para optimizar y conseguir el vector de thetas óptimo que ajuste a los vectores de datos dados a razón
# de un paso (alpha) definido y una n cantidad de iteraciones
# - Input:
#       - x: Vector de X (YA conteniendo la columna de X0 con "unos")
#       - y: Vector de Y
#       - thetas: Vector de thetas iniciales para el entrenamiento
#       - alpha: Learning Rate o paso para el ajuste de pesos
#       - iteraciones: épocas a ejecutar
# - Return :
#       - thetasGradiente: np.array de thetas óptimas
def gradienteDescendente(x, y, thetas, alpha, iteraciones)

# Función para calcular el costo dado un Vector de X, Y y thetas
# - Input:
#       - x: Vector de X (Conteniendo la columna de X0)
#       - y: Vector de Y
#       - theta: Vector de thetas a evaluar
```

```
# - Return :
#       - Costo calculado como: J(thetas) = (1/2m)(sumatoria(( f_hypothesis - y )**2))
def calcularCosto(x, y, thetas)
```

Ejemplo de Ejecución

A continuación se muestra un ejemplo de ejecución para describir la manera en la que se deben utilizar las funciones anteriormente mencionadas utilizando los datos de prueba brindados:

- En caso de querer leer un archivo con las entradas para construir los vectores se puede hacer uso del método readFile de la clase MLDev que recibe la ruta del archivo asumiendo que contiene encabezados para las columnas y que tiene un carácter con el cual se realiza la separación e interpretación del dataset.

```
mlObject = MLDev() # Instancia para el manejo de procesos de ML
path = "ex1data1.txt"

# Lectura de archivo mediante un caracter separador y llenado de vectores
mlObject.readFile(path, "\t")
```

  Para acceder a los vectores construidos se pueden utilizar las propiedades de la clase que contienen los arreglos de X (considerando la columna de X0 con “unos”) y de Y:

```
mlObject = MLDev() # Instancia para el manejo de procesos de ML
path = "ex1data1.txt"

# Lectura de archivo mediante un caracter separador y llenado de vectores
mlObject.readFile(path, "\t")

mlObject.X # Vector con X incluyendo x0
mlObject.Y # Vector con Etiquetas Y
```

- **Gradiente Descendente:** Para utilizar la función que calcula el gradiente descendente es necesario contar con un vector arreglado de X (Considerando la columna de x0), Y y thetas (En caso de haber construido los vectores con el método readFile anteriormente mencionado se pueden utilizar directamente los atributos que almacenan estos vectores en la estructura). Se debe indicar un paso alpha (0.01 utilizado como estándar) y un número definido de iteraciones/épocas durante las cuales se realizará la optimización. Al final la función retorna el vector thetas con los pesos actualizados y óptimos, además muestra una gráfica donde se muestra el costo a través de las épocas del entrenamiento para evaluar el aprendizaje del modelo. Para el ejemplo se utiliza un vector de thetas inicializado con valores “0”:

```
# Vector de thetas iniciales tamaño (n_variables, 1)
thetas = np.zeros((mlObject.X.shape[1], 1))

# Thetas óptimas resultantes del Entrenamiento en busca de thetas óptimas mediante gradiente descendente
thetas = gradienteDescendente(mlObject.X, mlObject.Y, thetas, 0.01, 1500)
print("\nThetas Final: \n", thetas)
```

Mostrando como output:

Thetas Final:
[[-3.63029144]
[1.16636235]]

El gráfico de costo a través de las épocas se visualiza de la siguiente manera:

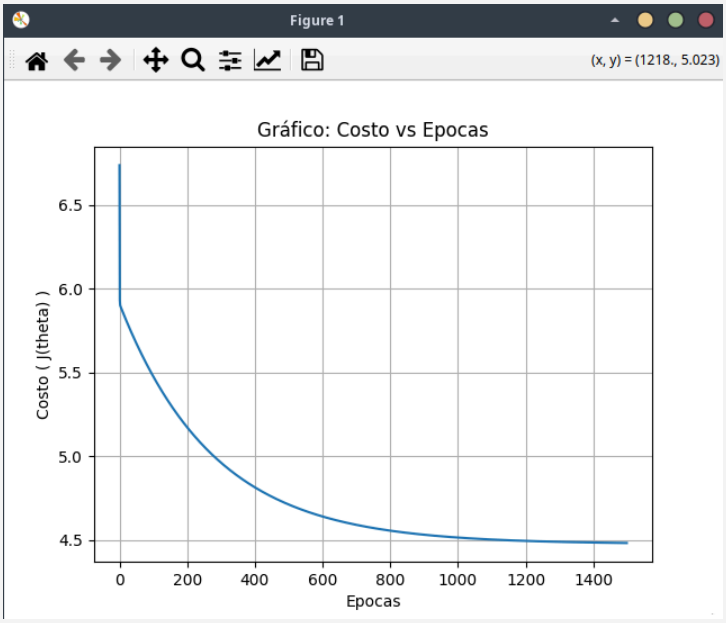

Figura 1: Ejemplo de visualización gráfica del costo a través de las épocas.

Para realizar una predicción con los valores thetas óptimos obtenidos se puede realizar de la siguiente manera:

```
print("\nPredicciones")
print(np.dot(np.array([1, 3.5]), thetas))
print(np.dot(np.array([1, 7]), thetas))
```

Mostrando el siguiente output:

Predicciones
[0.45197679]
[4.53424501]

- **graficaDatos:** Para utilizar la función que grafica la dispersión de los vectores de entrada X y Y, para además dibujar la recta ajustada con los valores óptimos resultantes para theta, es necesario colocar como parámetros el arreglo de X (Asumiendo que aún contiene la columna de x0 con "unos"), el arreglo de Y (De dimensiones (m, 1)) y el vector de thetas. La graficación se ejecuta de la siguiente manera:

```
# Graficación de dispersión con función ajustada visible
print("\nGraficando dispersión y recta ajustada con valores theta: \n", thetas)
graficaDatos(mlObject.X, mlObject.Y, thetas)
```

La gráfica se vería de la siguiente manera:

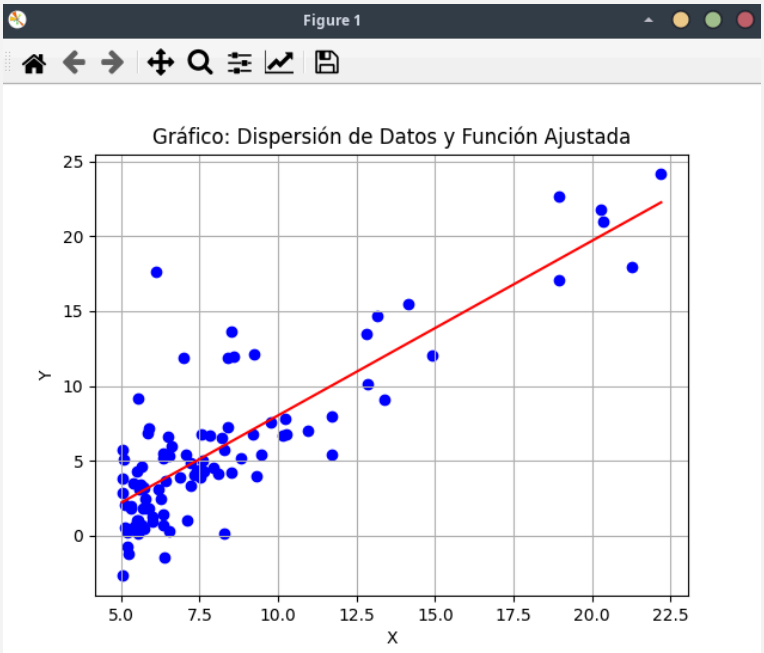

Figura 2: Ejemplo de visualización gráfica de dispersión de los datos y la recta ajustada mediante la regresión.

- **calcularCosto:** Para utilizar la función que calcula el costo como *J(thetas) = (1/2m)(sumatoria((f_hypothesis - y)**2))*, se debe colocar como parámetros un arreglo de X (Conteniendo la columna de x0 con "unos"), un arreglo Y y un vector de thetas respecto a las cuales se calculará el costo. Lo anterior se demuestra en el siguiente ejemplo:

```
# Ejemplo de cálculo de costo
print("\nEjemplo de calculo de costo:")
print(calcularCosto(mlObject.X, mlObject.Y, thetas))
```

El output se puede visualizar de la siguiente manera:

Ejemplo de cálculo de costo:

4.483388256587725

Conclusión

Finalmente, se puede decir que mediante la utilización de los procesos implementados anteriormente para ejecutar el entrenamiento de un modelo de regresión lineal simple utilizando la técnica de gradiente descendente para encontrar los valores óptimos para los pesos de la función propuesta, es posible realizar la predicción de las ganancias del foodtruck. Lo cual podemos visualizar en las predicciones realizadas, donde:

- **Población = 3.5**
- **Posibles Ganancias = 0.4519**

y:

- **Población = 7**
- **Posibles Ganancias = 4.5342**

Siendo que las predicciones realizadas anteriormente fueron calculadas utilizando los valores óptimos de theta obtenidos por el modelo:

Thetas Final : [[-3.63029144] [1.16636235]]